In Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP'98),
Mar 4-5, 1998, Clearwater Beach, FL, pp. 85-91

Low-Cost Pathways Towards Formal Methods Use

Martin S. Feather
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, MS 125-233
Pasadena CA 91109, USA
+1 818 354 1194
Martin.S.Feather@Jpl.Nasa.Gov

1. ABSTRACT

In current practice, formal methods are perceived as high-cost activities, and hence their use is recommended primarily for cases warranting the highest possible level of assurance. However, opportunities abound for beneficial applications of formal methods across a broad spectrum of cases, provided low-cost pathways towards their introduction and use can be identified. This premise is illustrated on a fragment of space vehicle requirements. Other researchers have studied fragments similar to this to illustrate various analysis techniques. Here it is shown that judicious choice of representation permits (some) formal analysis to be conducted immediately. Furthermore, this representation is made alluring by automatically generating textual and tabular representations from it. The net result is a low-cost (perhaps even cost-savings) approach to manipulating requirements of this nature, with the beneficial side effect of permitting formal analysis of those requirements at no extra cost.

1.1 Keywords

Analysis, flowcharts, formal methods, requirements, tabular representations, traceability.

2. INTRODUCTION

A typical scenario of formal methods application is to identify some tricky portion of a design whose correctness is critical to the project in question, expend time and effort to recast that design as a formal specification, and challenge that specification with increasingly sophisticated analysis and validation techniques. The kinds of operations performed on a formal specification might include type checking, simulation, symbolic evaluation, model checking or theorem proving. It is common practice to recommend this as a cost-effective approach on "important" applications, where presumably the cost of failure is high, thus justifying the expense of the approach. For example, the NASA formal methods guidebook [8] states, in the section on Cost Considerations, that: "… prudent advice to projects would be the following. In the context of a stable, controlled software process that includes an emphasis on quality assurance in the requirements phase, generate a formal specification for a core subset of important requirements. …"

The premise of this paper is that opportunities abound for beneficial applications of formal methods across a *broad* spectrum of cases, provided *low-cost* pathways towards their introduction and use can be identified. The premise is illustrated on a fragment of space vehicle requirements. Other researchers ([2] [9]) have studied fragments similar to this to illustrate various analysis techniques. Herein, it is shown how the judicious choice of representation of these requirements would allow (some) formal analysis to be conducted immediately, and at no extra cost. Furthermore, this representation is made alluring by showing how from it, textual and tabular representations of these requirements could be automatically generated. The net result is a low-cost (perhaps even cost-savings) approach to manipulating requirements of this nature, with the beneficial side effect of permitting immediate formal analysis of those requirements.

The paper is organized as follows: Section 3 introduces the example's requirements. Section 4 shows how to represent these requirements in a machine-manipulable fashion, thereby admitting to immediate and straightforward analysis. Section 5 discusses the advantages to multiple representational forms of requirements, as exemplified by the example's requirements documentation. Section 6 shows the automatic generation of textual and tabular forms of those requirements from the machine-manipulable form, thus making the adoption of the machine-manipulable form more alluring. Section 7 exhibits how traceability between the multiple requirements representations can be provided. Finally, section 8 concludes.

3. EXAMPLE - SPACE VEHICLE FDIR

The source of the example for this paper is a draft of software requirements for a space vehicle. The fragments of requirements concern the Bus Failure, Detection, Isolation and Recovery aspects, which concern handling of failures in communication buses.

The requirements document employs two representations: sets of paragraphs of structured text, and a flowchart-like depiction. A sample of the textual representation is shown in the paragraphs that follow, while the flowchart appears below (numbers have been appended to the flowchart nodes for reference within this paper):

RT Failure Detection/Skip requirements

While acting as the bus controller, the C&C MDM CSCI shall set the e,c,w, indicator identified in Table 3.2.16.3-I E,C,W BC/RT Event Definition, set failure status to "failed" if errors of CSE_RTSA messages occur in two consecutive processing frames, a backup RT is not available, the transaction errors are from only one RT, and:

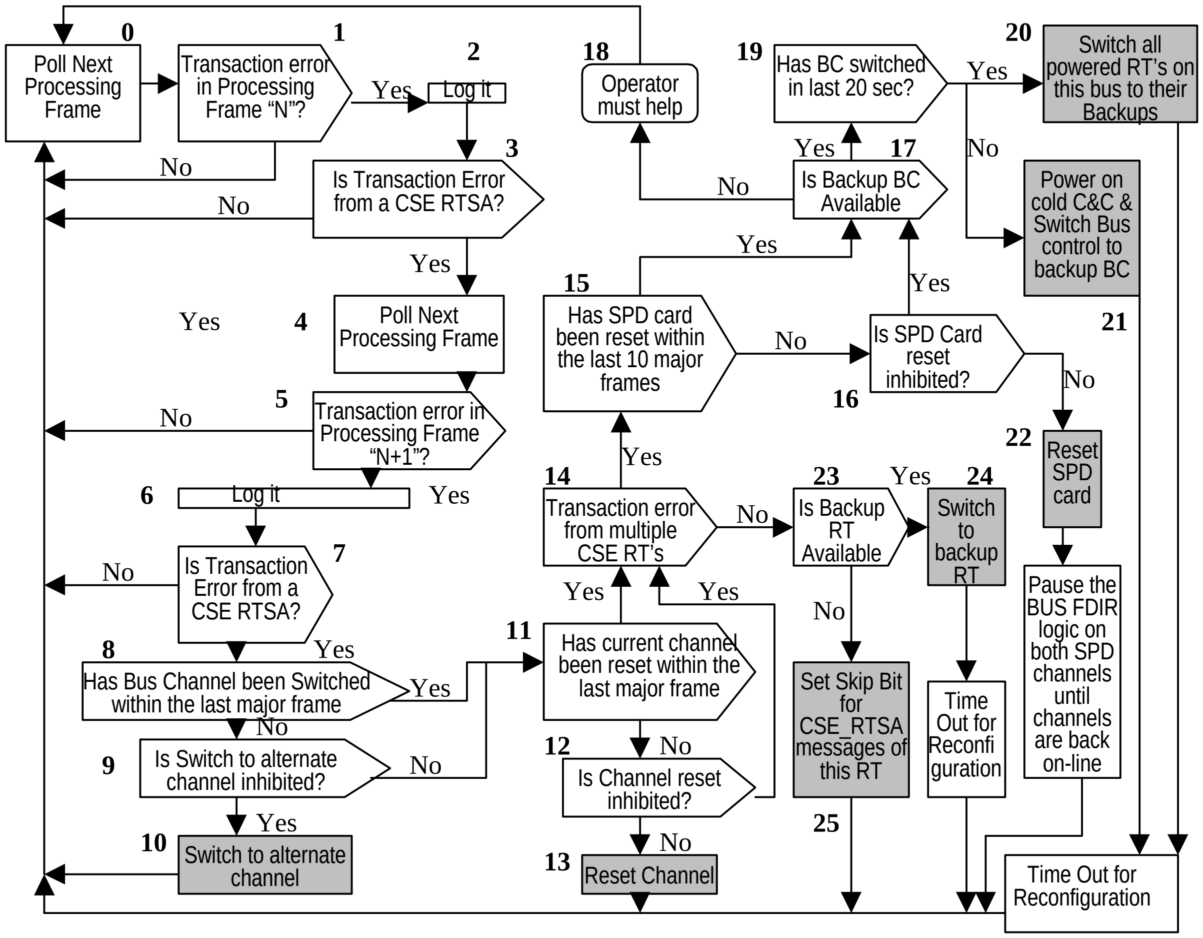
the SPD card has been reset within the last 100 seconds, the current bus channel has been reset within the last 10 seconds, and the bus channel has been switched in the last 10 seconds

the SPD card has not been reset within the last 100 seconds, the channel reset is inhibited, the current bus channel has not been reset within the last 10 seconds, and the bus channel has been switched in the last 10 seconds

the SPD card has not been reset within the last 100 seconds, the channel reset is inhibited, the current bus channel has been reset within the last 10 seconds, the alternate bus channel is not available, and the bus channel has not been switched in the last 10 seconds

the SPD card has not been reset within the last 100 seconds, the current bus channel has been reset within the last 10 seconds, the alternate bus channel is not available, and the bus channel has not been switched in the last 10 seconds

The requirements document contains several such sets of


0
Poll Next Processing Frame
1
Transaction error in Processing Frame "N"?
Yes
2
Log it
No
3
Is Transaction Error from a CSE RTSA?
No
Yes
4
Poll Next Processing Frame
Yes
5
Transaction error in Processing Frame "N+1"?
No
6
Log it
Yes
7
Is Transaction Error from a CSE RTSA?
No
Yes
8
Has Bus Channel been Switched within the last major frame
Yes
No
9
Is Switch to alternate channel inhibited?
No
Yes
10
Switch to alternate channel
11
Has current channel been reset within the last major frame
12
Is Channel reset inhibited?
No
13
Reset Channel
No
14
Transaction error from multiple CSE RT's
Yes
Yes
Yes
15
Has SPD card been reset within the last 10 major frames
No
16
Is SPD Card reset inhibited?
Yes
No
17
Is Backup BC Available
Yes
No
18
Operator must help
19
Has BC switched in last 20 sec?
Yes
No
20
Switch all powered RT's on this bus to their Backups
21
Power on cold C&C & Switch Bus control to backup BC
22
Reset SPD card
Pause the BUS FDIR logic on both SPD channels until channels are back on-line
23
Is Backup RT Available
Yes
No
24
Switch to backup RT
25
Set Skip Bit for CSE_RTSA messages of this RT
Time Out for Reconfiguration
Time Out for Reconfiguration

textual paragraphs, one set for each set of paths from node

0, "Poll Next Processing Frame", to a gray-colored action node of the flowchart. These total several pages. The set of paragraphs shown above correspond to the decision paths in the flowchart from node 0 to node 25, "Set Skip Bit for CSE_RTSA messages of this RT" (towards the lower right hand corner of the flowchart).

Both representations describe the decision process to follow to determine actions to take in the event of certain "transaction errors". Presumably this decision process should be identical in both representations. For reasons of conciseness perhaps, some of the textual representation's details on the queries to evaluate and actions to are not given in the flowchart. The textual descriptions spell out timing details (e.g., "last 100 seconds") whereas the flowchart employs somewhat more generic and/or terse descriptions (e.g., "10 major frames"). Nevertheless, these two representations provide alternative means of expression of the same overall decision process. For example, the first bulleted paragraph of the textual requirements states "CSE_RTA messages occur in two consecutive processing frames", corresponding to the "Yes" choices of nodes 1, 3, 5 and 7. Likewise, it states "a backup RT is not available", corresponding to the "No" choice of node 23, and "errors are from only one RT", corresponding to the "No" choice of node 14. Each of the four bulleted items captures one of the four pathways through the intermediate choice nodes 8,9,11 and 12. Because of the mismatch in level of detail of the two forms of description, this correspondence is sometimes a trifle obscure.

Section 6 will show how a structured textual style of description can easily be generated automatically from the flowchart representation. This suggests that it would suffice to develop and maintain a single representation (the flowchart), and generate the textual descriptions as needed, rather than construct them by hand. The next section discusses the advantages of making the flowchart the primary representation.

4. FORMAL ANALYSIS OPPORTUNITIES

The intent of this experiment is to illustrate a pathway via which formal methods can be introduced at little or no additional cost. The expression of the requirements information in the form of the flowchart nodes-and-arcs representation is the foundation of this pathway. This section considers simple formal analyses that can readily be conducted in terms of this flowchart representation.

Highly desirable properties of the bus channel management requirements are that they be *complete* (i.e., the requirements should state a course of action to take for every possible combination of circumstances) and *consistent* (i.e., for no possible combination of circumstances should the requirements state more than one course of action to take).

The textual representation of the requirements is not conducive to either of these kinds of analyses. Unfortunately, the fully detailed requirements of this system are expressed textually, while the flowchart is a simplified accompaniment. Hence, those conducting validation and verification efforts of actual requirements similar to these have had no choice but to work with the textual requirements. [2] reports such work, revealing:

- the difficulty inherent in unambiguously interpreting textual requirements,
- the virtue of a more formal representation, in particular, they employed a Leveson-style [4] tabular representation of the requirements' complex conditionals, and
- the use of a mechanical checker of the aforementioned completeness and consistency properties (also referred to as coverage and disjointness) - after initial experiments with PVS [7], they used SCR* [5] for this purpose.

Overall, this appears a somewhat time-consuming, and hence costly, V&V activity. In contrast, the use of the flowchart as the full-fledged representation considerably simplifies this whole process, because it permits easy analysis, both visually (at least for small flowcharts) and mechanically:

- Incompleteness would be manifest as missing arrows and/or nodes. For example, a node without an emerging arrow labeled "Yes" would indicate an unconsidered case. Similarly, an arrow that did not terminate on a node would indicate a case that, while considered, lacked a specific course of action. These are easy to detect by visual inspection on small flowcharts, or by a trivial mechanical check run over the internal representation.
- Inconsistency would be manifest as duplicated arrows. For example, a query node with two "Yes" branches emerging and leading to different destinations. Such inconsistency is also easy to detect by visual inspection and trivially mechanizable checks. Note that it is quite acceptable for different sets of conditions to lead to the same course of action, readily recognizable in the flowchart where multiple arrows point to the same node.

Small programs have been written to execute these, and a representative set of similar checks and queries (for example, the query of whether node 13 could possibly be reached [i.e., the reset channel action could be taken] if the "No" branch of node 8 has been taken [i.e., the Bus Channel has been switched within the last major frame]). The nodes-and-arcs representation of the flowchart was found to be highly conducive to this. In contrast, the English text is somewhat opaque to analysis. For example, it takes some patience to study the textual requirements sample given in section 3 enough to realize that it does *not* appear to match the flowchart! The text's first paragraph includes the clauses "…a backup RT is not available, the transaction errors are from only one RT…", obviously referring to node 23's "No" branch downstream of node 14's "No" branch. However, each of the bulleted items begins by referring to resets of the SPD card, i.e., node 15, which is downstream of node 14's "Yes" branch, rather than its "No" branch!

5. ADVANTAGES OF GENERATING THE STRUCTURED TEXTUAL REQUIREMENTS

The simple analyses of the previous section can be done at very low cost given a flowchart representation of requirements. However, the fragment of space vehicle requirements studied employed a flowchart as, seemingly, an adjunct to the more thorough textual requirements. Thus the approach being advocated here is intended to *suggest* opportunities for future requirements activities. This can become alluring if it can be done at no extra cost over current practice.

Observe that there are two representations of the same requirements, a flowchart, and structured textual paragraphs. Multiple representations are desirable to have, but to manually maintain them is an expensive and error-prone activity. The suggested approach is to generate one of the representations from the other.

Obvious reasons for the desirability of multiple representations include:

- different people may be comfortable with different representations;
- alternative representations may draw attention to different aspects of the requirements;
- alternative representations may be conducive to different forms of analysis;
- contractual obligations may necessitate delivery of the requirements in a particular representation, not necessarily the one that the requirements engineers would wish to work with.

In particular, natural language representations have almost universal appeal to human readers. Until people become adept with a formal notation, they will favor natural language descriptions over that notation. For example, [11] reports on the appeal of a specification paraphraser tool that produced English descriptions from a specification language, thus alleviating the new reader from the need to first learn the specification language's syntax. Since there will often be such reluctance to have to learn yet another formal notation, natural language is likely to retain its appeal as a means of expression. An advantage to having examples generated from the specific requirements fragments is that it provides a description of the meaning of the notation on the very example in which the reader has an interest.

Surprisingly, even when people *are* conversant with some formal notation, they may continue to benefit from a natural language description of requirements in addition to a formal language description of the same. An illustration of this is reported in [12] (follow-on work to the specification paraphraser). A second tool was developed, to generate English descriptions of execution traces (generated by a symbolic evaluator of the specification language). This second tool turned out to be helpful to people *familiar* with the specification language, as well as those unfamiliar. Swartout states the reason for this as being that: "… an English translation gives the specifier an alternate view of his specification which highlights some aspects of the specification which are easily overlooked in the formal Gist notation".

This effect would probably hold true of any pair of notations, whether or not one of them was natural language. Indeed, using the same notation, but reorganizing the presentation, could have the same beneficial effect of highlighting different aspects of a description.

The disadvantages of *manually* maintaining multiple representations include:

- the cost of creating multiple representations of the same requirements information;
- the cost of maintaining multiple representations as changes occur to the requirements;
- the need to ensure that the multiple representations indeed describe the same requirements information. (This can be circumvented by always giving priority to one representation's interpretation in the case of differences, but this is a far from satisfactory solution).

The generative approach, in which requirements are expressed once in one representation, and from which the other representation is automatically generated, yields the best of both worlds – multiple representations at minimal cost and error. The next section shows the feasibility of generating structured text from flowcharts, illustrated on the space vehicle requirements fragments.

6. AUTOMATIC GENERATION OF ALTERNATIVE REPRESENTATIONS

Generation of a (somewhat naive but passable) structured text representation from a representation of the flowchart is straightforward. As evidence of this a simple program, `flowtalk`, has been constructed to do such text generation from a nodes-and-arcs representation of the flowchart. In this experiment, the nodes-and-arcs representation has *not* been linked to a graphical portrayal of the flowchart, although to do so would be a small additional programming effort (or falls within the capability of existing tools).

An example of `flowtalk`'s output is shown on the next page. This text corresponds to the decision paths in the flowchart from node 0, the "Poll next processing frame" node in the upper left hand corner, to node 25, the "Set

Skip Bit for CSE_RTSA messages of this RT" node towards the lower right hand corner of the flowchart.

```
IF Poll next processing frame
AND Transaction error in processing
frame N?
THEN Log it
AND Is transaction error from a CSE
RTSA?
THEN Poll next processing frame
AND Transaction error in processing
frame N+1?
THEN Log it
AND Is transaction error from a CSE
RTSA?
AND
  (
  IF Has bus channel been switched
  within the last major frame?
  )
  OR
  (
  IF NOT Has bus channel been
  switched within the last major
  frame?
  AND NOT Is switch to alternate
  channel inhibited?
  )
AND
  (
  IF Has current channel been reset
  within the last major frame?
  )
  OR
  (
  IF NOT Has current channel been
  reset within the last major frame?
  AND Is channel reset inhibited?
  )
AND NOT Transaction error from
multiple CSE RTs?
AND NOT Is backup RT available?
THEN Set skip bit for CSE_RTSA
messages of this RT
```

The same text-generating capabilities can equally well be applied to present individual query results. For example, the query of paths from node 7 to node 11:

Flowchart-node-numbers: `((7 8 11) (7 8 9 11))`

Structured text:

```
IF Is transaction error from a CSE RTSA?
AND
  (
  IF Has bus channel been switched
  within the last major frame?
  )
  OR
  (
  IF NOT Has bus channel been switched
  within the last major frame?
  AND NOT Is switch to alternate channel
  inhibited?
  )
```

Parentheses and indentation are used to help disambiguate the precedence of logical expressions. In contrast, [2] found this aspect of hand-written English requirements (similar to those on which this experiment is based) particularly hard to disambiguate in this regard.

The generated form adheres more directly to the structure of the flowchart than does the original English statement of the requirements. For example, the flowchart's paths from node 0 to node 25 bifurcate at node 8, re-converge and then bifurcate again at node 11. In the original English statement, each of the four possible resulting paths (two alternatives through the first choice, times two alternatives through the second) is a separate bulleted item. In generating the English, it is straightforward to arrange to have it mirror the flowchart's structure - the result is the

```
  ( … )
  OR
  ( … )
AND
  ( … )
  OR
  ( … )
```

layout. Of course, if desired, it is also straightforward to arrange to generate a separate clause for each of the four paths.

In addition to generating text, `flowtalk` can also generate Leveson-style AND/OR tables for the conditional part of the decision paths from one node to another. The table generated for paths from node 0 to node 25 is shown below. This table shows the four combinations of conditions - the rows are the conditions involved, the four columns of T/F/. the combinations of those conditions' values for each case.

Transaction error in processing frame N?	T	T	T	T
Is transaction error from a CSE RTSA?	T	T	T	T
Transaction error in processing frame N+1?	T	T	T	T
Is transaction error from a CSE RTSA?	T	T	T	T
Has bus channel been switched within the last major frame?	T	T	F	F
Is switch to alternate channel inhibited?	.	.	F	F
Has current channel been reset within the last major frame?	T	F	T	F
Is channel reset inhibited?	.	T	.	T
Transaction error from multiple CSE RTs?	F	F	F	F
Is backup RT available?	F	F	F	F

This table is comparable to those produced by hand as reported in [2].

7. PROVIDING TRACEABILITY BETWEEN REPRESENTATIONS

There is one potential difficulty with this generative approach. A purely generated representation cannot be modified directly, for to do so would violate the correspondence between that derived representation and the source representation from which is derived. Instead, the corresponding modification must be made on the source representation. If generation were *two-way*, that is, given a pair of representations, one could be derived from the other, and vice-versa, then modification *could* be made to directly to either representation, and the other one re-generated. However, generation is often far easier to do in one direction than in the other.

Such is the case with flowcharts and their textual equivalents. It is easy to generate structured text from an arbitrary flowchart (as demonstrated by the program whose outputs have been shown), but hard to generate a flowchart from structured text, and also hard to constrain people to write structured text without straying from the dictates of the structuring scheme. Natural language is abundantly expressive, with consequent difficulties when it comes to its automated processing. Research successes on this front (e.g., [1], [3]) have tended to focus on the extraction of entity-relationship models, abstractions, data-dictionaries, and the like, but flow-of-control remains a challenge.

To ameliorate this problem, the `flowtalk` program provides the option to generate simple traceability information linking the two representations. When this option is chosen, traceability information is generated and embedded in to the text itself, so that it is easy to refer back to the portion(s) of the flowchart from which the text is derived. This is accomplished by labeling each of the nodes in the flowchart (which was already done in order to represent the flowchart), and embedding these labels into the generated text. A fragment of the generated structured text show earlier is repeated below, this time with the trace information included. The numbers in square parentheses correspond to the node numbers in our representation of the flowchart, and so can be used to trace back to the actual node in the flowchart from which that portion of text is derived.

```
AND [7] Is transaction error from a CSE
RTSA?
AND
  (
  IF [8] Has bus channel been switched
  within the last major frame?
  )
  OR
  (
  IF NOT [8] Has bus channel been
  switched within the last major frame?
  AND NOT [9] Is switch to alternate
  channel inhibited?
  )
AND
  (
  IF [11] Has current channel been reset
  within the last major frame?
  )
  OR
  (
  IF NOT [11] Has current channel been
reset within the last major frame?
  AND [12] Is channel reset inhibited?
  )
AND NOT [14] Transaction error from
multiple CSE RTs?
AND NOT [23] Is backup RT available?
THEN [25] Set skip bit for CSE_RTSA
messages of this RT
```

When generating Leveson-style tables `flowtalk` is able to insert traceability information in a similar manner, as shown at the bottom of this page.

8. CONCLUSIONS

This paper has illustrated how a set of real-world requirements could be represented in a machine-manipulable form, and thereby become immediately amenable to easy analysis. This is based on one of the representational forms already employed by the requirements documentation, with the added advantage that the second representational form is automatically generable from the first. A popular tabular representation is also automatically generable. Furthermore, simple traceability information can also be generated between the

```
[1]Transaction error in processing frame N?                    T T T T
[3]Is transaction error from a CSE RTSA?                       T T T T
[5]Transaction error in processing frame N+1?                  T T T T
[7]Is transaction error from a CSE RTSA?                       T T T T
[8]Has bus channel been switched within the last major frame?  T T F F
[9]Is switch to alternate channel inhibited?                   . . F F
[11]Has current channel been reset within the last major frame? T F T F
[12]Is channel reset inhibited?                                . T . T
[14]Transaction error from multiple CSE RTs?                   F F F F
[23]Is backup RT available?                                    F F F F
```

representational forms. The net result is a plausible means to introduce (simple) formal methods techniques into practice, in a way that dovetails with current styles of expression. This is intended to be alluring by virtue of its ability to automatically generate the second representational form, i.e., might save some effort even without taking into account the increased analyzability.

An important aspect that has not been considered is scalability. Flowcharts much larger than a single page cease to be convenient or understandable. As with any approach, some form of information hiding and hierarchical structuring is required to handle large sets of requirements. It is interesting to note that the existing space vehicle requirements do already take some small steps in this direction, by employing terse descriptions in the flowchart representation, and by relegating to further tables action details that do not themselves affect the decision process.

Flowtalk is just a simple program, far less sophisticated than the tools commonly brought to bear in formal methods activities. Likewise, its generation of textual descriptions is simplistic (see, for example, [10] for much more impressive work in this area), and its approach to maintaining multiple viewpoints is also straightforward (again, for comparison, see [6]). Yet, by application to a judiciously selected representation of the requirements, this approach is capable of performing a variety of analyses that otherwise appear cumbersome to conduct. The primary message this exercise is intended to convey is that low-cost pathways towards the introduction of formal methods do exist, and work in this area promises to yield increasing opportunities for beneficial applications of formal methods.

9. ACKNOWLEDGMENTS

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space administration. Funding was provided under NASA's Code Q Software Program Center Initiative UPN #323-08.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

10. REFERENCES

[1] Ambriola & Gervazi: Processing Natural Language Requirements. In *Proceedings of the Twelfth IEEE International Automated Software Engineering Conference*: 36-45, November 1997.

[2] Easterbrook & Callahan: Formal Methods for V&V of partial specifications: An experience report. In Proceedings *of the Third IEEE International Symposium On Requirements Engineering*: 160-168, January 1997.

[3] Goldin & Berry: AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in Requirements. In *Automated Software Engineering* 4(4): 375-412, October 1997.

[4] Leveson, Heimdahl, Hildreth & Reese: Requirements specification for process-control systems. In *IEEE Transactions on Software Engineering*: 20(9), September 1994.

[5] Heitmeyer, Bull, Gasarch & Labaw: SCR* A toolset for specifying and analyzing requirements. In *Tenth Annual Conference on Computer Assurance (COMPASS '95)*: 109-122, June 1995.

[6] Johnson, Feather & Harris: Representation and Presentation of Requirements Knowledge. In *IEEE Transactions on Software* Engineering 18(10): 853-869, October 1992.

[7] Owre, Rushby & Shankar: Analysing tabular and state-transition specifications in PVS. Technical Report CSL-95-12, Computer Science Laboratory, SRI International, 1995.

[8] NASA: Formal Methods Specification and Verification Guidebook for Software and Computer Systems. Volume I: Planning and Technology Insertion, NASA-GB-002-95 Release 1.0, NASA, Washington, D.C., July 1995.

[9] Russo, Nuseibeh & Kramer. Restructuring Requirements Specifications for Managing Inconsistency Analysis and Change: A Case Study. To appear in *Proceedings of* the *International Conference on Requirements Engineering*, 1998.

[10] Salek, Sorenson, Tremblay & Punshon. The REVIEW System: From Formal Specifications to Natural Language. In *Proceedings of* the *First International Conference on Requirements Engineering*: 220-229, April 1994.

[11] Swartout. GIST English generator. In *AAAI-82 Proceedings of the National Conference on Artificial Intelligence*: 404-409, 1982.

[12] Swartout. The GIST behavior explainer. In *AAAI-83 Proceedings of* the *National Conference on Artificial Intelligence*: 1983.